Implementing a Dialog System in Python

Ingmar van der Geest	Elfia Bezou Vrakatseli	Markos Polos	Marik Westerhof
UU	UU	UU	UU
Utrecht, Netherlands	Utrecht, Netherlands	Utrecht, Netherlands	Utrecht, Netherlands
Team 17	Team 17	Team 17	Team 17

ABSTRACT
UPDATED—7 October 2019. This paper constitutes a detailed description of a project in the Human-Computer Interaction (HCI) field. The aim of our project was the implementation of a goal-based conversational agent in the restaurant recommendation domain, which performs a text-based dialog with users. The user informs the system about their preferences for a restaurant, according to which the system finds a suitable restaurant in a database and provides additional details if necessary. The backend of this agent uses a LSTM-trained machine classifier in a python-based environment to classify and process the user input and return an answer, resulting in said restaurant recommendation. Classification accuracy of the model reached up to 81.1% but did not use a pre-existing lexical corpus. The resulting dialog system attempted to confirm most of the data explicitly (i.e. 'Do you want to look for a cheap place to eat Chinese food?") to prevent any wrong recommendations and performed reasonably well.

Author Keywords
Human-computer interaction; dialogue system; restaurant recommendation.

INTRODUCTION
A widely used A.I. application are the conversational agents, or dialog systems. These computer systems are programmed to communicate with humans (users) in natural language (written, oral, or both), and fall into two general categories; task-oriented dialog agents and chatbots. Our project is the implementation of the former.

Task-oriented dialog agents, or goal-based conversational agents, are designed for particular tasks, and are created in such a way that they are capable of having short conversations in order to gather information from the user. This information is then used by the system to complete the given task, e.g., to book a flight, or to recommend a restaurant. One example of task-oriented dialog agents are digital assistants, e.g., Siri or Alexa.

We have developed a task-oriented dialog agent, called Ambrosia, which interacts with users, in order to give them a restaurant recommendation. Our system is programmed in Python 3 and it is terminal/command line based. It is designed to ask for certain preferences (area, price range, food type), and look for a recommendation based on these preferences. After the system finds suitable restaurants in a database, it suggests one to the user. Once the restaurant has been accepted by them, Ambrosia can provide additional information about the restaurant (e.g., phone number, address, postal code).

The creation of Ambrosia was based on an existing dataset with annotated dialogs in the restaurant domain and it consisted of multiple stages. Firstly, the dialogs were extracted and modeled, and a state transition diagram was then created based on them. The procedure concerning the data is described in detail in the following section, Data. Moreover, we made use of the dialog acts in the dataset. The importance of assigning dialog parts is not to be underestimated, since a dialog act maps an utterance to an action. There were 15 dialog acts in our dataset; *ack*, *affirm*, *bye*, *confirm*, *deny*, *hello*, *inform*, *negate*, *null*, *repeat*, *reqalts*, *reqmore*, *request*, *restart*, *thankyou*.

Two baseline systems were, then, implemented; a rule based one, that uses rules to classify the user utterance based on keyword matching (e.g., an utterance containing the word "hello" is classified with the dialog act hello). The other one assigns labels according to the label distribution of dialog acts in the dataset (e.g., if 50% of the utterances are labeled as *hello*, the system will predict *hello* 50% of the time, independently of the utterance). A machine learning classifier was created to be trained on the dialog act data, using Long-Short Term Memory (LSTM) regression to categorize user utterances (more details can be found in the machine learning section).

The combination of the aforementioned steps resulted in the implementation of a dialog manager that makes use of the transition diagram and the utterance classification to perform the dialogs. Ambrosia is also equipped with some configurable variations in the dialog model; these variations help for the evaluation part, specifically they were used to implement a human user evaluation experiment. In the dialog manager section, a detailed description of our transition dialog function can be found.

DATA
In order to create our system, we made use of an existing dataset; 3235 text-based, annotated dialogs were used. These dialogs consist of exchange of utterances between the system and each user, regarding the restaurant domain, and they were collected using speech recognition.

More specifically, the dialogs that were used consist of information exchanges between the system and users. The first part of each dialog consists of information gathering by the system; it asks the user for certain preferences (area, price range, food type), in order to proceed to suggesting. After the preferences are stated by the user, the system makes a restaurant suggestion, if the preferences are sufficient. If the preferences given by the user do not suffice, the system asks for supplementary information. After the suggestion is made by the system, the user either asks for further information for the selected restaurant or requests an alternative. The system, then, either goes on to providing information about the suggested restaurant or suggests a different one (if there is one), respectively. The additional information is either the restaurant's phone number, its address, or its postal code.

As mentioned above, the dialogs were collected using automatic speech recognition, thus the language of the dialogs incorporates features of oral speech, which is always less precise, compared to written language. Firstly, the level of formality of the utterances is low, as expected in an oral transcription, and self-referencing pronouns (e.g., "I") are very common. Moreover, the user utterances consist of short sentences, words with few syllables, and they include phrases such as "okay um". Furthermore, the grammar of the utterances is also influenced; sometimes the sentences are incomplete or start with "and" or "but".

The fact that the language used for training our system reflects the properties of oral speech, has an impact the system's performance, since the system we trained will be used for written communication. Differences between oral and written communication influence the system's performance; for instance, in written communication, phrases such as "okay um" will be much more rare, dropping the frequency of the ack act very low. Another difference is that in written communication we expect more spelling mistakes by the users (e.g., when a user is in a hurry, or perhaps not a native speaker), but we also expect the frequency of completely irrelevant words (which can occur as a result of an incorrect transcription) to drop. Finally, an oral communication contains much more noise compared to a written one.

The data was extracted from two data sets (two separate files in JSON format); the label and the log data, where the utterances are organized by turn index. The former consists of the data concerning the user, whereas the latter consists of the data concerning the system. As far as the label data is concerned, the data has a standard specification for each dialog; there is a unique caller id, a session id, and the dialog turns as a list of label turn objects, which run in parallel to the log data.

Before using the data, the dialogs were converted to lowercase letters. Doing that, for both training and testing, results in increasing the classifier's accuracy. To this end, all user input was converted to lowercase, both when manually entering test cases and during user input for the final dialog system.

Finally, a program was developed so as to recreate the dialogs as an immediate exchange of utterances, as shown in Figure 1. These dialogs were then modeled, and a state transition diagram was then created based on these dialogs. See Figure 3 for this diagram.

```
system: Hello , welcome to the Cambridge restaurant system?
        You can ask for restaurants by area , price range
        or food type . How may I help you?
user: cheap spanish restaurant
system: la raza serves spanish food in the cheap price range
user: address
system: Sure , la raza is on 4 - 6 Rose Crescent
user: phone number
system: The phone number of la raza is 01223 464550 .
user: thank you good bye
```

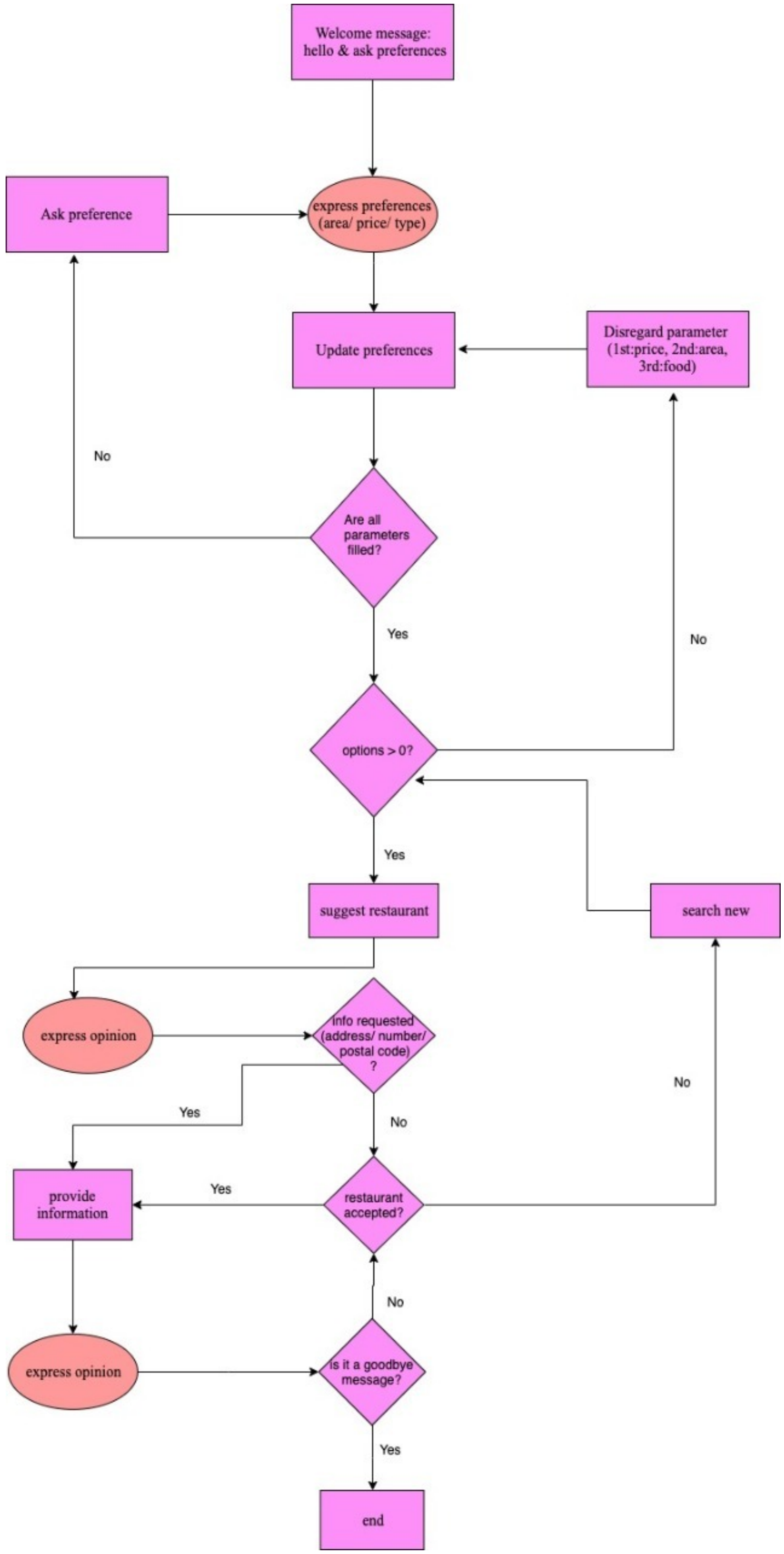
Figure 1: Conversation snippet 1

In this dialog snippet we can observe how the system relates to some dialog states. Firstly, there is a welcome message by the system, followed by the user's preferences (price range = cheap, food type = Spanish). In this dialog, the preferences stated by the user suffice for the system to look for a suggestion, and thus, the system replies with a restaurant recommendation. In line 4, the user asks for additional information, i.e. the restaurant's address, and the system responds with the requested information in the next line. In the sixth line the user asks for another piece of information concerning the suggested restaurant, i.e. the phone number, which the system proceeds on providing in the following line. In the final line, the user gives a goodbye message, which indicates the end of the dialog.

```
system: Hello , welcome to the Cambridge restaurant system? You can
       ask for restaurants by area , price range or food type .
       How may I help you?
user: moderately priced restaurant that serves creative food
system: I'm sorry but there is no restaurant serving creative food
user: how about modern european
system: efes restaurant is in the moderate price range
user: phone number
system: The phone number of efes restaurant is 01223 500005 .
user: thank you good bye
```

Figure 2: Conversation snippet 2

In this dialog snippet in Figure 2**,** a more complicated dialog is illustrated. Firstly, there is a welcome message by the system, followed by the user's preferences (price range=moderately priced, food type=creative food). In this dialog, the preferences stated by the user suffice for the system to look for a suggestion, however, there were no restaurants matching these criteria. Thus, the message returned by the system is an apologetic message, which requests for a change in the preferences. The user, then, changes the food type preference to "modern European". The system updates the preferences and returns a new suggestion, while implicitly confirming the other parameter (price range) at the same time. In the sixth line, the user requests some information about the suggested restaurant, specifically the phone number, and the system proceeds to providing the requested information. In the final line, the user gives a goodbye message, which indicates the end of the dialog.


Welcome message: hello & ask preferences
Ask preference
express preferences (area/ price/ type)
Update preferences
Disregard parameter (1st:price, 2nd:area, 3rd:food)
No
Are all parameters filled?
Yes
No
options > 0?
Yes
suggest restaurant
search new
express opinion
info requested (address/ number/ postal code) ?
No
Yes
No
provide information
Yes
restaurant accepted?
No
express opinion
is it a goodbye message?
Yes
end

MACHINE LEARNING

The Machine Learning syntax classification was mostly created using the NLTK and Keras modules, and follows the pipeline displayed in Figure 4. Firstly, the data that was previously extracted from the SQL files and put into simple text files is imported using the Pandas module, and subsequently split in labels and text. Blank rows are then filtered out, and all text is converted to lowercase prior to the tokenization of the text. This tokenization splits the strings of sentences into strings of words. Considering a bag-of-words approach is used in accordance with the assignment, saving the n-grams of word strings is not necessary. After tokenization, stop words and non-alphanumeric characters are also taken out, and words are stemmed and lemmatized. Word strings are categorized as nouns, adjectives, verbs, and adverbs, using pre trained part-of-speech tagging from the NLTK module. At this stage, the data sets are now ready-to-use train and test corpora.

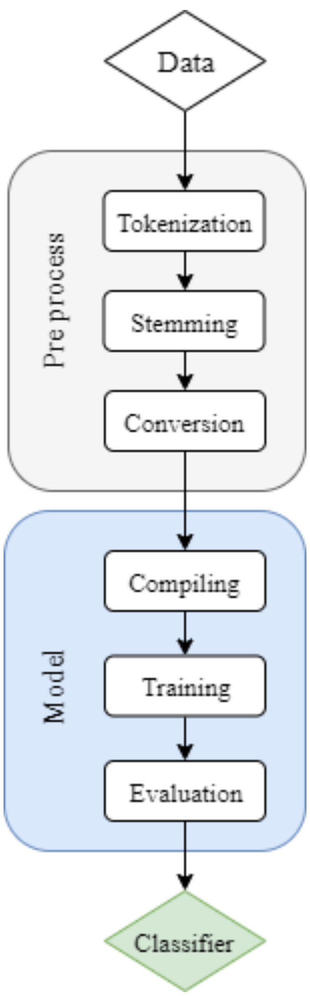


Figure 4: Machine Learning Pipeline

A brief analysis of the label distribution of the corpora showed that the test data is very unevenly distributed; as shown in Figure 5, the test set contained few instances of *hello*, *ack*, *confirm*, and *deny*. In fact, it did not contain a single instance of three labels, namely *reqmore*, *request* and *thankyou*, making it impossible to fully test our model for classifying. The uneven distribution of both train and test data may cause the model to classify rather biased towards more frequently occurring labels.

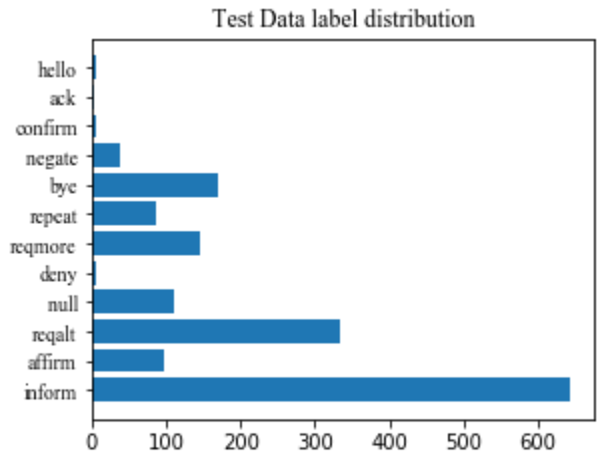


Figure 5: Label distribution for test data

After preprocessing, the corpus is used by a compiled sequential model from the Keras module containing Long-Short Term Memory (LSTM) to train itself over five epochs with 64-long batch sizes, using otherwise standard settings. The Keras evaluation function returned an average accuracy of 81.1%, with a loss of 0.373. The highest test accuracy score was already obtained in the third epoch, however, as can be seen in Figure 6**.** This shows that the subsequent epochs likely lead to overfitting of the model to the train set.

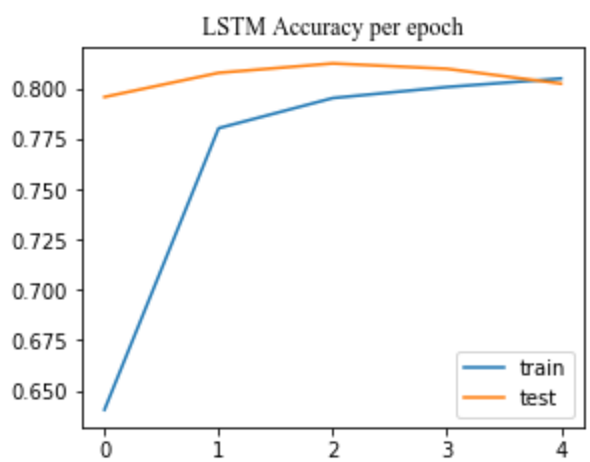


Figure 6: Accuracy performance per iteration

Some classification errors also appeared after evaluating the model. Longer sentences containing more information were much more prone to misclassification, especially if they contained multiple types at once, such as a confirmation and a question at the same time. The utterance ‘okay’ (and any derivations thereof) as part of a longer sentence were often classified as null or, alternatively, as any seemingly random labels. Thus, the model would proceed to mislabel the entire sentence as a result from a tilt by the term 'okay'. Other issues arose with the usage of any apostrophe in a sentence: this model would misclassify any utterance as a negation simply for having this character present in the first place, as if its existence equals a grammatical negation. For additional suggested improvements, see the discussion section.

DIALOG SYSTEM

For the dialog manager, a state transition function was built based on the aforementioned dialog state transition diagram (Figure 3). This function's role is to determine the actions of the system, based on the user's utterance. This function has the current dialog state and current user utterance as input and the next dialog state (with associated system utterance) as output.

To this end, it needs to determine the dialog act of the user's utterance, and then transition to the next state accordingly. Thus, we made use of the dialog acts resulting from the classifier we built. To give an example, if the user utterance has been classified as *goodbye*, then the system goes to the *bye* state, regardless of the state we are in. Alternatively, if at the start of the dialog the user utterance is classified as *inform*, then the next dialog state is *confirm*, where information is gathered, and then, confirmation is requested to be given by the user, regarding the stated preferences.

To be able to transition to a new state, various transition state functions were built, e.g. `state_hello()`, `state_request()`, `state_confirm()`, `state_inform()`, `state_bye()`. For each of these, a conditional if-then-else clause determines to what state the system goes next, based on the dialog act of the user's latest utterance. Moreover, a function with the name `extracting_preferences()` is used to extract the user's preferences based on regular expressions combined with Levenshtein's distance.

The database of all the restaurants was preprocessed by transferring it into a Python list of dictionaries, where each dictionary represents one restaurant. These dictionaries contained 7 keys, holding all of the information on that restaurant. The json ontology of all possible preferences (area, price range, food type) was imported through the json module, and the sections for *area*, *price range*, *food type* and were added to global variables as Python lists.

The `extracting_preferences()` function will look through the ontology first, finding a match of the user's preference in terms of area, price range, and food type or a complement with a Levenshtein edit distance equal or smaller than 1 (as implemented in the python-Levenshtein library). The Levenshtein distance is used to map values to the closest domain term in case an exact match is not found. Levenshtein distance is defined as the number of characters that need to be inserted, deleted, or substituted (i.e., the number of edit operations) to convert a string into another string. For instance, the utterance "I want a restaurant serving Greec food" contains a spelling mistake, and by using Levenshtein distance, we are able to deduct the intended user's utterance. According to Levenshtein distance, the misspelled word "Greec" has a distance 1 from "Greek". Choosing the match with the smallest distance from the possible alternatives results in the correct value as the preference (Yujian & Liu, 2007).

Finally, a lookup function retrieves suitable restaurant suggestions from the CSV database matching the preferences as extracted in the implemented algorithm. The `get_matches()` function in our system selects a list of restaurants fulfilling the user's criteria, or indicating that no restaurants were available or more specifications were needed.

DISCUSSION

The current dialog transition method is very linear, in the sense that once a certain state has been reached then the only way to return to a previous state is a "hard" reset. For example, once the user has received a suggestion, the only way to get a suggestion based on different preferences is by resetting all their preferences. Furthermore, the system will always ask for all preferences to be specified, even if based on 2 parameters there are only a very limited number of restaurants left. This can be improved easily by letting the system return a suggestion the moment the number of possible restaurants is below a certain amount.

As for the dialog act classification, three approaches were implemented. Firstly, a baseline for rule-based dialog act classification system was built. There were multiple limitations to this approach, since it is a rather stiff approach, which fails if a sentence does not identically match the regular expressions that were used to build the system. The lack of flexibility of this approach reduces its efficiency, however, this approach serves as a reference point for our neural classifier.

The second approach, the baseline that assigns a label act depending on the label distribution, also faced great challenges, since the degree of randomness for the classification of one utterance was very high.

The third approach consists of training a LSTM classifier on text-based dialogs, which used natural language, as described in the machine learning section. One limitation of this classifier is that it was only trained on texts collected using speech recognition, whereas it is used to classify data of written communication. Therefore, the classifier is trained on data which make use of a slightly different language, both grammatically and lexically (e.g., utterances such as "okay um" are nonexistent in texts of written communication). This system worked reasonably well with its reported classification accuracy of 81.1% but could have contained several improvements.

In order to significantly improve accuracy results for the latter method, a lexical corpus from the NLTK database could have been used for training the model; instead, this classification algorithm used only the vocabulary of the utterances present in the training data. While there existed no out-of-vocabulary words in this manner, the model's understanding of words might be rather limited, and would react erratically upon receiving a prompt with an unknown word. Furthermore, extensive research into choosing an optimal regression algorithm could improve both the results and the time efficiency. For results, experimentation with Keras' Convolutional LSTM might have procured improved results and us suggested for further research. Secondly, selecting Gated Recurrent Unit (GRU) over LSTM could have provided significantly increased efficiency with similar results, especially for its strong compatibility to run on GPUs rather than CPUs with the Tensorflow_gpu backend (Chung et al., 2014). These shortened training durations may also have provided more time for extensive parameter optimization. However, as LSTM seems to outperform GRU with texts consisting of longer sentences (Kaiser & Sutskever, 2015), chances of this regression algorithm providing a higher score than LSTM are assumed to be very small.

TEAM CONTRIBUTIONS

Team contributions were as following. Our team followed the weekly suggested deadlines by the program and started working ahead on the report around the time of the 1c already.

	Task	**Person**	**Time**
1a	Display dialogs in terminal	Markos	5
	Convert source JSONs to text file	Marik	5
	Create transition diagram	Elfia/Ingmar	4
1b	Produce text file of utterance	Markos	7
	Keyword matching baseline	Elfia	8
	Dialog act distribution baseline	Ingmar	4
	Machine Learning classifier	Marik	20
1c	State transition functions	Ingmar	5
	User preference statements	Markos	12
	Lookup functions	Markos	13
	Sentence generation	Elfia/Ingmar	10
	Dialog model configurations	Ingmar/Markos	6
	General debugging	Ingmar	15
Report	Abstract/ Introduction	Elfia/Marik	4
	Data	Elfia	7
	Machine Learning	Marik	8
	Dialog system	Elfia/Ingmar	3
	Discussion	Elfia	5
	Formatting	Marik	3

REFERENCES

1. Chung et al. (2014) *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.* Obtained from https://arxiv.org/pdf/1412.3555v1.pdf
2. Kaiser & Sutskever (2015) *Neural GPUs Learn Algorithms*. Obtained from https://arxiv.org/pdf/1511.08228.pdf
3. Yuijan & Liu (2007) A Normalized Levenshtein Distance Metric, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 29*(6),* 1091-1095. Obtain from https://ieeexplore.ieee.org/abstract/document/4160958